A Joint Exponential Mechanism For Differentially Private Top-\(k\)

Jennifer Gillenwater 1  Matthew Joseph 1  Andrés Muñoz Medina 1  Mónica Ribero 1 2

Abstract

We present a differentially private algorithm for releasing the sequence of \(k\) elements with the highest counts from a data domain of \(d\) elements. The algorithm is a “joint” instance of the exponential mechanism, and its output space consists of all \(O(d^k)\) length-\(k\) sequences. Our main contribution is a method to sample this exponential mechanism in time \(O(dk \log(k) + d \log(d))\) and space \(O(dk)\). Experiments show that this approach outperforms existing pure differential privacy methods and improves upon even approximate differential privacy methods for moderate \(k\).

1. Introduction

Top-\(k\) is the problem of identifying the ordered sequence of \(k\) items with the highest counts from a data domain of \(d\) items. This basic problem arises in machine learning tasks such as recommender systems, basket market analysis, and language learning. To solve these problems while guaranteeing privacy to the individuals contributing data, several works have studied top-\(k\) under the additional constraint of differential privacy (DP) (Dwork et al., 2006). Differential privacy guarantees that publishing the \(k\) identified elements reveals only a controlled amount of information about the users who contributed data to the item counts.

The best known DP algorithm for top-\(k\) is the “peeling” mechanism (Bhaskar et al., 2010; Durfee & Rogers, 2019), which applies a DP subroutine for selecting the highest counts from a data domain of \(d\) elements. This leads to the baseline peeling mechanisms in our experiments, each using the best known composition methods: the approximate DP baseline uses the exponential mechanism analyzed via concentrated differential privacy (CDP) composition (Dwork & Rothblum, 2016; Bun & Steinke, 2016), and the pure DP baseline uses the permute-and-flip mechanism with basic composition. The approximate DP variant takes time \(O(d + k \log(k))\) and the pure variant takes time \(O(dk)\). Both require space \(O(d)\).

1.1. Our Contributions

We construct an instance of the exponential mechanism that chooses directly from sequences of \(k\) items. Unlike the peeling mechanism, this approach does not use composition. Past work has used this style of “joint” exponential mechanism to privately and efficiently estimate: 1-way marginals under \(\ell_\infty\) error (Steinke & Ullman, 2015), password frequency lists (Blocki et al., 2016), and quantiles (Gillenwater et al., 2021). However, it is not obvious how to extend any of these to top-\(k\) selection.

Naive implementation of a joint exponential mechanism, whose output space is all sequences of \(k\) items, requires enumerating all \(O(d^k)\) such sequences. This is impractical even for modest values of \(d\) and \(k\). As with previous work on joint exponential mechanisms, our main contribution is an equivalent efficient sampling method.

Theorem 1.1 (Informal version of Theorem 3.9). There is a joint exponential mechanism for \(\varepsilon\)-DP top-\(k\) that takes time \(O(dk \log(k) + d \log(d))\) and space \(O(dk)\).

While it is straightforward to prove a utility guarantee for this mechanism (Theorem 3.3), our main argument for this joint approach is empirical, as asymptotic guarantees often obscure markedly different performance in practice. Experiments show that the joint exponential mechanism offers the strongest performance among pure differential privacy mechanisms and even outperforms approximate differential privacy mechanisms when \(k\) is not large (Section 4).

1.2. Related Work

Private top-\(k\) was first studied in the context of frequent itemset mining, where each user contributes a set of items, and the goal is to find common subsets (Bhaskar et al.,
We start by formally describing the top-
k upper bound signed maximum error (Theorem 3.3),
which the true
Notation. For dataset
s
of k items \( TOP-k(D) = \arg \min_{S=\{s_1, \ldots, s_k\}} \ell(S). \)
Note that each user contributes at most one to each item count but may contribute to arbitrarily many items. This captures many natural settings. For example, a user is unlikely to review the same movie more than once, but they are likely to review multiple movies. In general, we describe dataset \( D \) by the vector of counts for its domain, \( D = (c_1, \ldots, c_d). \)

Past work has also studied several other algorithms for DP top-
k. Laplace noise has been used for pure and approximate DP (Ding et al., 2019; Qiao et al., 2021). Additionally, for pure DP, the Gamma mechanism for releasing private counts (Theorem 4.1, (Steinke & Ullman, 2015)) can be applied. Our experiments found that peeling mechanisms dominate these approaches, so we omit them, but implementations of all three appear in the supplementary material. Finally, we note that Durfee & Rogers (2019) study the problem of private top-
k when the number of items \( d \) is prohibitively large and \( O(d) \) runtime is impractical. We instead focus on the setting where \( O(d) \) runtime is acceptable.

A few lower bounds are relevant to private top-
k. One naive approach is to simply privately estimate all \( d \) item counts and return the top \( k \) from those noisy estimates. However, each count has expected error \( \Omega(d/\varepsilon) \) for pure DP (Theorem 1.1, (Hardt & Talwar, 2010)). Similarly, Bun et al. (2014) (Corollary 3.4) construct a distribution over databases such that, over the randomness of the database and the mechanism, each count has expected error \( \Omega(\sqrt{d}/\varepsilon) \) for approximate DP. Top-
k approaches aim to replace this dependence on \( d \) with a dependence on \( k \). Bafna & Ullman (2017) and Steinke & Ullman (2017) prove lower bounds for what we call \( k \)-relative error, the maximum amount by which the true \( k \)th count exceeds any of the estimated top-
k counts (see Section 2 for details). Respectively, they prove \( \Omega(k \log(d)) \) and \( \Omega(\sqrt{k \log(d)}) \) sample complexity lower bounds for “small” and “large” \( k \)-relative error. In both cases, the peeling mechanism provides a tight upper bound. We upper bound signed maximum error (Theorem 3.3), but our paper more generally departs from these works by focusing on empirical performance.

2. Preliminaries

Notation. \([m] = \{1, \ldots, m\}\). Given a vector \( v \) in dimension-
\( d \) and indices \( i < j \), \( v_{ij} \) denotes coordinates \( i, \ldots, j \) of \( v \); given a sequence \( S \), \( v_S \) denotes coordinates \( v_i \) for \( i \in S \).

We start by formally describing the top-
k problem.

Definition 2.1. Let \( \mathcal{X} \) be a data domain of \( d \) items. In the case of the top-
k problem, there are \( n \) users and each user \( i \in [n] \) has an associated vector \( x_i \in \{0,1\}^d \).

For dataset \( D = \{x_1, \ldots, x_n\} \), let \( c_j = \sum_{i=1}^n x_{ij} \) denote the count of item \( j \in [d] \), and let \( (c_1, \ldots, c_d) \) denote the counts in nonincreasing order. Given sequence of indices \( S \), let \( c_S \) denote the corresponding sequence of counts. Given sequence loss function \( \ell \), the goal is to output a sequence \( S \) of \( k \) items \( TOP-k(D) = \arg \min_{S=\{s_1, \ldots, s_k\}} \ell(S). \)

The specific choice of error may be tailored to the data analyst’s goals: \( \ell_{\infty} \) error suits an analyst who wishes to minimize the worst error of any of the top \( k \) counts; \( \ell_1 \) error is appropriate for an analyst who views a sequence of slightly inaccurate counts as equivalent to one highly inaccurate count; and \( k \)-relative error may be best when the analyst prioritizes a “sound” sequence where no count is much lower than the true \( k \)th count. Note that, while \( k \)-relative error has been featured in past theoretical results on private top-
k selection, it is the most lenient error metric. For example, given \( d \) items with counts \( 100, 1, 1, \ldots, 1 \) and \( k = 2 \), any sequence of items obtains optimal \( k \)-relative error, and in general \( \ell_{k-rel}(c_S) = \max_{i \in [k]} (c_k - c_{s_i}). \)

Next, we cover privacy prerequisites. Differential privacy guarantees that adding or removing a single input data point can only change an algorithm’s output distribution by a carefully controlled amount.

Definition 2.3 (Dwork et al. (2006)). Datasets \( D, D' \in \mathcal{X}^* \) are neighbors (denoted \( D \sim D' \)) if \( D' \) can be obtained from \( D \) by adding or removing a data point \( x \). Mechanism \( M: \mathcal{X}^* \rightarrow Y \) is \((\varepsilon, \delta)\)-differentially private (DP) if, for any two neighboring datasets \( D \sim D' \) in \( \mathcal{X}^* \), and any \( S \subseteq Y \), it holds that \( \mathbb{P}[M(D) \in S] \leq e^{\varepsilon}\mathbb{P}[M(D') \in S] + \delta \). If \( \delta = 0 \), it is \( \varepsilon \)-DP.

One especially flexible differentially private algorithm is the exponential mechanism. Given some utility function over outputs, the exponential mechanism samples high-utility outputs with higher probability than low-utility outputs.

Definition 2.4 (McSherry & Talwar (2007); Dwork & Roth (2014)). Given utility function \( u: \mathcal{X}^* \times \mathcal{O} \rightarrow \mathbb{R} \) with \( \ell_1 \) sensitivity \( \Delta(u) = \max_{D \sim D', o \in \mathcal{O}} |u(D, o) - u(D', o)| \), the exponential mechanism \( M \) has output distribution \( \mathbb{P}[M(D) = o] \propto \exp\left( \frac{\varepsilon u(D, o)}{2\Delta(u)} \right) \),
A tighter analysis is possible for certain utility functions.

**Lemma 2.5** (McSherry & Talwar (2007)). The exponential mechanism is \(\varepsilon\)-DP.

A tighter analysis is possible for certain utility functions.

**Definition 2.6.** A utility function \(u\) is monotonic (in the dataset) if, for every dataset \(D\) and output \(o\), for any neighboring datasets \(D'\) that results from adding some data point to \(D\), \(u(D, o) \leq u(D', o)\).

When the utility function is monotonic, the factor of 2 in the exponential mechanism’s output distribution can be removed. This is because the factor of 2 is only necessary when scores for different outputs move in opposite directions between neighboring datasets.

**Lemma 2.7** (McSherry & Talwar (2007)). Given monotonic utility function \(u\), the exponential mechanism is \(\frac{\varepsilon}{2}\)-DP.

One cost of the exponential mechanism’s generality is that its definition provides no guidance for efficiently executing the sampling step. As subsequent sections demonstrate, this is sometimes the main technical hurdle to applying it.

### 3. A Joint Exponential Mechanism for Top-\(k\)

Our application of the exponential mechanism employs a utility function \(u^*\) measuring the largest difference in counts between the true counts and candidate counts. Let \((c_1, \ldots, c_d)\) be the item counts in nonincreasing order. For candidate sequence of items \(S = (s_1, \ldots, s_k)\), we define

\[
u^*(D, S) = \begin{cases} 
-\max_{i \in [k]} (c_i - c_{s_i}) & \text{if } s_1, \ldots, s_k \text{ are distinct.} \\
-\infty & \text{otherwise}
\end{cases}
\]

\(u^*\) thus assigns the highest possible score, 0, to the true sequence of top \(k\) counts, and increasingly negative scores to sequences with smaller counts. Sequences with repeated items have score \(-\infty\) and are never output.

**Discussion of \(u^*\).** A natural alternative to \(u^*\) would replace \(-\max_{i \in [k]} (c_i - c_{s_i})\) with \(-\max_{i \in [k]} |c_i - c_{s_i}| = -\|c_i - c_{s_i}\|_{\infty}\). Call this alternative \(u'\). In addition to being expressible as a simple norm, \(u'\) also corresponds exactly to the number of user additions or removals sufficient to make \(S\) the true top-\(k\) sequence\(^1\). However, \(u^*\) has two key advantages over \(u'\). First, \(u^*\) admits an efficient sampling mechanism. Second, \(u'\) favors sequences that omit high-count items entirely over sequences that include them in the wrong order. For example, suppose we have a dataset \(D\) consisting of 10 items with counts 100, 90, \ldots, 10. If we want the top \(k\) = 5 items, we will consider sequences such as \(S_1 = (1, 3, 4, 5, 2)\) and \(S_2 = (1, 3, 4, 5, 6)\). These have identical value according to \(u^*: u^*(D, S_1) = -10 = u^*(D, S_2)\). But according to \(u'\), \(S_1\) scores much worse than \(S_2:\) \(u'(D, S_1) = -30 < -10 = u'(D, S_2)\). This conflicts with the ultimate goal of identifying the highest-count items; \(S_1\) contains item 2 (count 90), while \(S_2\) replaces it with item 6 (count 50).\(^2\) We now show that \(u^*\) also has low sensitivity.

**Lemma 3.1.** \(\Delta(u^*) = 1\).

**Proof.** First, any sequence with utility \(-\infty\) has that utility on every dataset. Turning to sequences of distinct elements \((s_1, \ldots, s_k)\), adding a user does not decrease any count, and increases a count by at most one. Furthermore, while the top \(k\) items may change, none of the top-\(k\) counts decrease, and each increases by at most one. It follows that each \(c_i - c_{s_i}\) either stays the same, decreases by one, or increases by one.

A similar analysis holds when a user is removed. \(\Box\)

We call the instance of the exponential mechanism with utility \(u^*\) JOINT. Its privacy is immediate from Lemma 2.5.

**Theorem 3.2.** JOINT is \(\varepsilon\)-DP.

A utility guarantee for JOINT is also immediate from the generic utility guarantee for the exponential mechanism. The (short) proof appears in Appendix A.

**Theorem 3.3.** Let \(c_1, \ldots, c_d\) denote the true top-\(k\) counts for dataset \(D\), and let \(\tilde{c}_1, \ldots, \tilde{c}_d\) denote those output by JOINT. With probability at least 99/100,

\[
\max_i (c_i - \tilde{c}_i) \leq \frac{2(k \ln(d) + 5)}{\varepsilon}.
\]

Naive sampling of JOINT requires computing \(O(d^k)\) output probabilities. The next subsection describes a sampling algorithm that only takes time \(O(dk \log(k) + d \log(d))\).

### 3.1. Efficiently Sampling JOINT

The key observation is that, while there are \(O(d^k)\) possible output sequences, a given instance \(u^*(D, \cdot)\) has only \(dk\) possible values. This is because each score takes the form \(-(c_i - c_j)\) for some \(i \in [k]\) and \(j \in [d]\). Our algorithm will therefore proceed as follows:

\(^1\)The idea of a utility function based on dataset distances has appeared in the DP literature under several names (Johnson & Shmatikov, 2013; Asi & Duchi, 2020; Medina & Gillenwater, 2020) but has not been applied to top-\(k\) selection.

\(^2\)The standard \(\ell_\infty\) loss metric shares this flaw: \(\ell(c_S) = \max_{i \in [k]} |c_i - c_S|\) may therefore be a reasonable loss metric for future top-\(k\) work. Nonetheless, past work uses \(\ell_\infty\) error, and we did not observe large differences between the two empirically, so we use \(\ell_\infty\) in our experiments as well.
1. For each of the $O(dk)$ utilities $U_{ij} = -(c_i - c_j)$, count the number $m(U_{ij})$ of sequences $S$ with score $u^*(S) = U_{ij}$.

2. Sample a utility $U_{ij}$ from the distribution defined by

$$P[U_{ij}] \propto m(U_{ij}) \exp\left(\frac{z U_{ij}}{2}\right). \quad (1)$$

3. From the space of all sequences that have the selected utility $U_{ij}$, sample a sequence uniformly at random.

This outline makes one oversimplification: instead of counting the number of sequences for each of $O(dk)$ (possibly non-distinct) integral utility values, the actual sampling algorithm will instead work with exactly $dk$ distinct non-integral utility values. Nonetheless, the output distribution will be exactly that of the exponential mechanism described at the beginning of the section.

### 3.1.1. Counting the Number of Sequences

Define $k \times d$ matrix $\tilde{U}$ by $\tilde{U}_{ij} = -(c_i - c_j) - z_{ij}$ where $z_{ij}$ is a small term in $(0, 1/2)$ that ensures distinctness,

$$z_{ij} = \frac{d(k - i) + j}{2dk}.$$ 

Several useful properties of $\tilde{U}$ are stated in Lemma 3.4.

**Lemma 3.4.** Given $\tilde{U}$ defined above, 1) each row of $\tilde{U}$ is decreasing, 2) each column of $\tilde{U}$ is increasing, and 3) the elements of $\tilde{U}$ are distinct.

**Proof.** Fix some row $i$. By definition, $c_1 \geq c_2 \geq \cdots \geq c_d$, so $-(c_i - c_1) \geq \cdots \geq -(c_i - c_d)$. The $z_{ij}$ terms also increase with $j$, so each row of $\tilde{U}$ is decreasing. By similar logic, each column of $\tilde{U}$ is increasing.

Finally, note that since $i \in [k]$ and $j \in [d]$, the $z$ terms are

$$\frac{dk}{2dk}, \frac{dk - 1}{2dk}, \ldots, \frac{2}{2dk}, \frac{1}{2dk}$$

and thus are $dk$ distinct values in $[0, 1/2]$. Since any two count differences $-(c_{i_1} - c_{j_1})$ and $-(c_{i_2} - c_{j_2})$ are either identical or at least 1 apart, claim 3) follows.

We now count “sequences through $\tilde{U}$”. Each sequence $(s_1, \ldots, s_k)$ consists of $k$ values from $[d]$, one from each row of $\tilde{U}$, and its score is $\min_{i \in [k]} \tilde{U}_{isi}$, or $-\infty$ if the $k$ values are not distinct. For each $i \in [k]$ and $j \in [d]$, define $\tilde{m}(\tilde{U}_{ij})$ to be the number of sequences through $\tilde{U}$ with distinct elements and score exactly $\tilde{U}_{ij}$. $\tilde{m}$ and $\tilde{U}$ are useful because of the following connection to $m(U_{ij})$, the quantities necessary to sample from the distribution in Equation 1:

**Lemma 3.5.** For any $i \in [k]$ and $j \in [d]$, let $A_{ij} = \{\tilde{U}_{ij'} \mid \tilde{U}_{ij'} = U_{ij}\}$. Then $m(U_{ij}) = \sum_{i' \in A_{ij}} \tilde{m}(\tilde{U}_{ij'})$.

**Proof.** Each $z \in (0,1/2]$, so $A_{ij}$ is exactly the collection of $\tilde{U}_{ij'} = -(c_{i'} - c_{j'}) - z_{ij'}$ where $c_{i'} - c_{j'} = c_i - c_j$.

The problem thus reduces to computing the $\tilde{m}$ values. For each row $r \in [k]$ and utility $\tilde{U}_{ij} \in \tilde{U}$, define

$$t_r(\tilde{U}_{ij}) = \max\{j' \in [d] \mid \tilde{U}_{ij'} \geq \tilde{U}_{ij}\}.$$ 

Useful simple properties of these $t_r$ values appear below.

**Lemma 3.6.** Fix some $\tilde{U}_{ij} \in \tilde{U}$. Then 1) $t_r(\tilde{U}_{ij})$ is non-decreasing in $r$, 2) if sequence $S = (s_1, \ldots, s_k)$ has score $\tilde{U}_{ij}$, then for all $r \in [k]$, $s_r \leq t_r(\tilde{U}_{ij})$ and 3) there exists a sequence $S = (s_1, \ldots, s_k)$ of distinct elements with score $\tilde{U}_{ij}$ if and only if $t_r(\tilde{U}_{ij}) \geq r$ for all $r$.

**Proof.** The first two properties follow directly from Lemma 3.4. For property 3) let $S = (s_1, \ldots, s_k)$ satisfy the conditions of the lemma and assume $t_r(\tilde{U}_{ij}) < r$ for some $r$. By properties 1) and 2), for all $r' \leq r$, $s_{r'} \leq t_{r'}(\tilde{U}_{ij}) \leq t_r(\tilde{U}_{ij}) < r$. But that implies $S$ contains $r$ distinct numbers less than $r$, which is a contradiction. In the other direction, suppose $t_r(\tilde{U}_{ij}) \geq r$ for all $r$. Define $S = (s_1, \ldots, s_k)$ where $s_r = r$ for $r < i$ and $s_i = j$. Since $t_r(\tilde{U}_{ij}) \geq i$, $j \geq i$. Therefore $[t_{i+1}(\tilde{U}_{ij})] - \{s_1, \ldots, s_i\}$ contains at least one option for $s_{i+1}$, $[t_{i+1}(\tilde{U}_{ij})] - \{s_1, \ldots, s_i\}$ contains at least one option for $s_{i+2}$, and so on. The resulting $S$ has distinct elements and score $\tilde{U}_{ij}$.

The following lemma connects the $t_r$ and $\tilde{m}$ values.

**Lemma 3.7.** Given entry $\tilde{U}_{ij}$ of $\tilde{U}$, define $n_r = \max(t_r(\tilde{U}_{ij}) - (r - 1), 0)$. Then $\tilde{m}(\tilde{U}_{ij}) = \prod_{r \neq i} n_r$.

**Proof.** By Lemma 3.6 we know the statement is true for $U_{ij}$ such that $\tilde{m}(U_{ij}) = 0$. If $\prod_{r \neq i} n_r > 0$ then any sequence with score $\tilde{U}_{ij}$ consists of distinct elements $S = (s_1, \ldots, s_k)$ such that $s_i = j$ and for all $r \neq i$, $s_r \in [t_r(\tilde{U}_{ij})]$ (otherwise $S$ has score less than $\tilde{U}_{ij}$). The number of such sequences is $\prod_{r \neq i} n_r (t_r(\tilde{U}_{ij}) - (r - 1))$.

A naive solution thus computes all $dk^2$ of the $t_r$ values, then uses them to compute the $\tilde{m}$ values. We can avoid this by observing that, if we sort the values of $\tilde{U}$, then adjacent values in the sorted order have almost identical $t_r$.

**Lemma 3.8.** Let $\tilde{U}_{(1)}, \ldots, \tilde{U}_{(dk)}$ denote the entries of $\tilde{U}$ sorted in decreasing order. For each $\tilde{U}_{(a)}$, let $r(a)$ denote its row index in $\tilde{U}$. Then: 1) for each $a \in [dk - 1]$, $t_{r(a+1)}(\tilde{U}_{(a+1)}) = t_{r(a+1)}(\tilde{U}_{(a)}) + 1$, and 2) for $r' \neq r(a + 1)$, $t_{r'}(\tilde{U}_{(a+1)}) = t_{r'}(\tilde{U}_{(a)})$. 
Proof. For 1), assume \( t_r(a+1) (\tilde{U}(a+1)) > t_r(a+1) (\tilde{U}(a)) + 1 \).
Then we can define \( j \) such that \( t_r(a+1) (\tilde{U}(a)) < j < t_r(a+1) (\tilde{U}(a+1)) \), which implies that \( \tilde{U}(a) > \tilde{U}(a+1) \). This contradicts \( \tilde{U}(a) \) and \( \tilde{U}(a+1) \) being adjacent in sorted order. For 2), assume there exists \( r' \neq r(a+1) \)
such that \( t_r'(\tilde{U}(a+1)) > t_r'(\tilde{U}(a)) \). (If we instead assumed \( t_r'(\tilde{U}(a+1)) < t_r'(\tilde{U}(a)) \), this would contradict the sorting order of \( U \) and the definition of \( t_r \)). This implies that \( \tilde{U}(a) > \tilde{U}(a+1) \), which again contradicts \( \tilde{U}(a) \) and \( \tilde{U}(a+1) \) being adjacent in sorted order.

According to the above lemma, we can compute all of the \( \tilde{m} \) as follows. First, sort the entries of \( \tilde{U} \), recording the row and column indices of each \( \tilde{U}(a) \) as \( r(a) \) and \( c(a) \). Then, create a vector storing the \( t \) values for \( \tilde{U}(1) \). These can be combined into \( \tilde{m}(\tilde{U}(1)) \) using Lemma 3.7. If \( \tilde{m}(\tilde{U}(1)) \) is non-zero, then we can get \( \tilde{m}(\tilde{U}(2)) \) simply by rescaling; according to Lemma 3.8, adding one to entry \( t_r(\tilde{U}) \) gives the new vector of \( t \)’s, so only one term in the formula for \( \tilde{m} \) changes in going from \( \tilde{U}(1) \) to \( \tilde{U}(2) \). We can thus compute each \( \tilde{m}(\tilde{U}(a)) \) in constant time, and compute all \( \tilde{m} \) values in time \( O(dk \log(k)) \).

3.1.2. Sampling a Utility

Given the \( \tilde{m} \) values above, we sample from a slightly different distribution than the one defined in Equation 1. The new distribution is,

\[
P \left[ \tilde{U}_{ij} \right] \propto \tilde{m}(\tilde{U}_{ij}) \exp \left( \frac{\varepsilon \tilde{U}_{ij}}{2} \right).
\]

When the \( \tilde{m} \) are large, sampling can be done in a numerically stable manner using logarithmic quantities (see, e.g., Appendix A.6 of Medina & Gillenwater (2020)).

3.1.3. Sampling a Sequence

After sampling \( \tilde{U}_{ij} \) from Equation 2, we sample a sequence of item indices uniformly at random from the collection of sequences with score \( \tilde{U}_{ij} \). The sample fixes \( s_i = j \). To sample the remaining \( k-1 \) items, we sample \( s_1 \) uniformly at random from \( \{ t_1(\tilde{U}_{ij}) \} \setminus \{ j \} \), \( s_2 \) from \( \{ t_2(\tilde{U}_{ij}) \} \setminus \{ s_1, j \} \), and so on. Lemma 3.6 guarantees that this process never attempts to sample from an empty set.

3.1.4. Overall Algorithm

JOINT’s overall guarantees and pseudocode appear below.

**Theorem 3.9.** JOINT samples a sequence from the exponential mechanism with utility \( u^* \) in time \( O(dk \log(k)) \) and space \( O(dk) \).

**Proof.** We sketch here, deferring details to Appendix B.

---

**Algorithm 1 Efficiently sampling JOINT**

1. **Input:** Vector of item counts \( c_1, \ldots, c_d \), number of items to estimate \( k \), privacy parameter \( \varepsilon \)
2. Sort and relabel items so \( c_1 \geq c_2 \geq \cdots \geq c_d \)
3. Construct matrix \( \tilde{U} \) by \( \tilde{U}_{ij} = -(c_i - c_j) - \frac{dk - i + j}{2dk} \)
4. Sort \( \tilde{U} \) in decreasing order to get \( \tilde{U}_{(1)}, \ldots, \tilde{U}_{(dk)} \), storing the (row, column) of each \( \tilde{U}_{(a)} \) as \( (r(a), c(a)) \)
5. Initialize \( n_1, n_2, \ldots, n_k \leftarrow 0 \)
6. Initialize set of non-zero \( n_1, N \leftarrow \emptyset \)
7. Initialize \( b \leftarrow 0 \)
8. for \( a = 1, \ldots, dk \) do
9. \( n_{r(a)} \leftarrow c(a) - r(a) - 1 \)
10. \( N \leftarrow N \cup \{ r(a) \} \)
11. if \( |N| = k \) then break
12. Set \( \tilde{m}(\tilde{U}(a)) \leftarrow 0 \), and set \( b \leftarrow a \)
13. Set \( p \leftarrow \prod_{r \in [k]} n_r \)
14. Compute \( \hat{m}(U^{(b+1)}) \leftarrow p/n_{r(a)} \)
15. for \( a = b + 1, \ldots, dk \) do
16. Set \( p \leftarrow p/n_{r(a)} \)
17. Compute \( \hat{m}(U^{(a)}) \leftarrow p \)
18. Update \( n_{r(a)} \leftarrow n_{r(a)} + 1 \)
19. Update \( p \leftarrow p/n_{r(a)} \)
20. Sample a utility \( U^{(a)} \) using Equation 2
21. Initialize size-\( k \) output vector \( s \) with \( s_i \leftarrow j \)
22. for index \( i' = 1, 2, \ldots, i-1, i+1, \ldots, k \) do
23. Compute \( t_{i'}(U^{(a)}) \) by iterating through row \( i' \) of \( U \)
24. Sample \( s_{i'} \) uniformly from \( \{ t_{i'}(U^{(a)}) \} \setminus \{ j, s_1, s_2, \ldots, s_{i'-1} \} \)
25. **Output:** Vector of item indices \( s \)

**Privacy:** By Lemma 3.5, to get \( m \) it suffices to compute \( \tilde{m} \), and by Lemma 3.7 it suffices to compute the \( t \) values. A score sampled from Equation 2 may be non-integral; taking its ceiling produces a utility \( \tilde{U}_{ij} = -(c_i - c_j) \), with the desired distribution from Equation 1.

**Runtime and space:** Referring to Algorithm 1, line 2 takes time \( O(d \log(d)) \) and space \( O(d) \). Line 3 takes time and space \( O(dk) \). Line 4 takes time \( O(dk \log(k)) \) and space \( O(dk) \); since each row of \( U \) is already decreasing, we can use \( k \)-way merging (Knuth, 1997) instead of naive sorting. All remaining lines require \( O(dk) \) time and space.

**JOINT** has the same guarantees (Theorem 3.2, Theorem 3.3) as the exponential mechanism described at the beginning of this section, since its output distribution is identical.

---

Note that even if items \( i_1 \) and \( i_2 \) have identical counts, they may have differing sequence counts, \( \tilde{m}(U_{i_1}) \neq \tilde{m}(U_{i_2}) \). The sampling in the loop on Line 22 implicitly makes up for this difference. See the full privacy proof in Appendix B.
4. Experiments

Our experiments compare the peeling and joint mechanisms across several real-world datasets using the error metrics from Definition 2.2. All datasets and experiment code are public (Google, 2022). As described in Section 1.2, we only present the best pure and approximate DP baselines. Other methods are available in the experiment code. For completeness, example error plots featuring all methods appear in Appendix C.3.

4.1. Comparison Methods

4.1.1. Pure DP Peeling Mechanism

We start with the pure DP variant, denoted PNF-P. It uses \( k \frac{\varepsilon}{\varepsilon'} \)-DP applications of the permute-and-flip mechanism, which uses the exponential mechanism under basic composition (Theorem 2 (McKenna & Sheldon, 2020)). We use the equivalent exponential noise formulation (Ding et al., 2021), where the exponential distribution \( \exp (\lambda) \) is defined over \( x \in \mathbb{R} \) by

\[
\mathbb{P} [x; \lambda] = 1_{x \geq 0} \cdot \lambda \cdot \exp (-\lambda x).
\]

(3)

Its pseudocode appears in Algorithm 2. We omit the factor of 2 in the exponential distribution scale because the count utility function is monotonic (see Definition 2.6 and Remark 1 of McKenna & Sheldon (2020)).

Algorithm 2 PNF-Peel, pure DP peeling mechanism

1: Input: Vector of item counts \( c_1, \ldots, c_d \), number of items to estimate \( k \), privacy parameter \( \varepsilon \)
2: Initialize set of available items \( A \leftarrow [d] \)
3: Initialize empty size-\( k \) output vector \( s \)
4: for \( j \in [k] \) do
5: for \( i \in A \) do
6: Draw exponential noise \( \eta \sim \exp (\varepsilon/k) \)
7: Compute noisy count \( \tilde{c}_i \leftarrow c_i + \eta \)
8: Set \( i^* \leftarrow \arg \max_{i \in A} \tilde{c}_i \)
9: Record chosen index, \( s[j] \leftarrow i^* \)
10: Remove chosen index, \( A \leftarrow A \setminus \{i^*\} \)
11: Output: Vector of item indices \( s \)

Lemma 4.1. PNF-Peel is \( \varepsilon \)-DP.

4.1.2. Approximate DP Peeling Mechanism

The approximate DP variant instead uses \( k \varepsilon' \)-DP applications of the exponential mechanism. We do this because the exponential mechanism admits a CDP analysis that takes advantage of its bounded-range property for stronger composition; a similar analysis for permute-and-flip is not known.

We use the Gumbel-noise variant of the peeling mechanism (Durfee & Rogers, 2019). This adds Gumbel noise to each raw count and outputs the sequence of item indices with the \( k \) highest noisy counts. The Gumbel distribution \( \text{Gumbel} (\beta) \) is defined over \( x \in \mathbb{R} \) by

\[
\mathbb{P} [x; \beta] = \frac{1}{\beta} \cdot \exp \left( -\frac{x}{\beta} - e^{-x/\beta} \right).
\]

(4)

and the resulting pseudocode appears in Algorithm 3.

Algorithm 3 CDP-Peel, approx DP peeling mechanism

1: Input: Vector of item counts \( c_1, \ldots, c_d \), number of items to estimate \( k \), privacy parameter \( \varepsilon' \)
2: for \( i \in [d] \) do
3: Draw Gumbel noise \( \eta \sim \text{Gumbel} (k/\varepsilon') \)
4: Compute noisy count \( \tilde{c}_i \leftarrow c_i + \eta \)
5: Output: Ordered sequence of the \( k \) item indices with the highest noisy counts

By Lemma 4.2 in Durfee & Rogers (2019), CDP-Peel has the same output distribution as repeatedly applying the exponential mechanism and is \( \varepsilon' \)-DP. A tighter analysis is possible using CDP. While an \( \varepsilon \)-DP algorithm is always \( \frac{\varepsilon}{\varepsilon'} \)-CDP, an \( \varepsilon \)-DP invocation of the exponential mechanism satisfies a stronger \( \frac{\varepsilon}{\varepsilon'} \)-CDP guarantee (Lemmas 3.2 and 3.4 (Cesar & Rogers, 2021)). Combining this with a generic conversion from CDP to approximate DP (Proposition 1.3 (Bun & Steinke, 2016)) yields the following privacy guarantee:

Lemma 4.2. CDP-Peel is \((\varepsilon, \delta)\)-DP for any \( \delta > 0 \) and

\[
\varepsilon = \frac{k\varepsilon'^2}{8} + 2\varepsilon' \sqrt{\frac{k \log (1/\delta)}{8}}.
\]

All of our approximate DP guarantees for CDP-Peel use Lemma 4.2.

4.2. Datasets

We use six datasets: Books (Soumik, 2019) (11,000+ Goodreads books with review counts), Foods (McAuley, 2014) (166,000+ Amazon foods with review counts), Games (Tamber, 2016) (5,000+ Steam games with purchase counts), Movies (Harper & Konstan, 2015) (62,000+ Movies with rating counts), News (Fernandes et al., 2015) (40,000+ Mashable articles with share counts), and Tweets (Bin Tareaf, 2017) (52,000+ Tweets with like counts). For each dataset, it is reasonable to assume that one person contributes \( \leq 1 \) to each count, but may also contribute to many counts. Histograms of item counts appear in Appendix C.1. A more relevant quantity here is the gaps between counts of the top \( k \) items (Figure 1, leftmost column). As we’ll see, JOINT performs best on datasets where gaps are relatively large (Books, Movies, News, and Tweets).

4.3. Results

The experiments evaluate error across the three mechanisms, six datasets, and three error metrics. For each mechanism,
Figure 1. Note that all plots have a logarithmic $y$-axis and quantities are padded by 1 to avoid discontinuities on the logarithmic $y$-axis. **Left column**: Count differences $c_k - c_{k+1}$ for each dataset. **Center column**: $\ell_\infty$ error. **Right column**: $\ell_1$ error.
the center line plots the median error from 50 trials (padded
by 1 to avoid discontinuities on the logarithmic y-axis), and
the shaded region spans the 25th to 75th percentiles. We
use \( k = 5, 15, \ldots, 195 \) with 1-DP instances of J
OINT and PNF-PEEL and \((1, 10^{-6})\)-DP instances of CDP-PEEL. Due
to the weakness of the \( k \)-relative error metric, and for the
sake of space in the figure, we relegate its discussion to
Appendix C.2.

4.3.1. \( \ell_\infty \) ERROR

J OINT’s performance is strongest for \( \ell_\infty \) error (Figure 1,
center column). This effect is particularly pronounced on
the Books, Movies, News, and Tweets datasets. This is
because these datasets have large gaps between the top \( k 
\) counts (Figure 1, leftmost column), which results in large
gaps between the scores that JOINT assigns to optimal and
suboptimal sequences. These large gaps enable JOINT to
obtain much stronger performance than the baseline pure DP
algorithm, PNF-PEEL, and to beat even the approximate DP
CDP-PEEL for a wide range of \( k \). In contrast, small gaps
reduce this effect on Foods and Games. On these datasets,
JOINT slightly improves on PNF-PEEL overall, and only
improves on CDP-PEEL for roughly \( k \leq 30 \).

\[ \text{Figure 2. } \ell_\infty \text{ index error plots. Note the logarithmic y-axis.} \]

The \( \ell_\infty \) metric also features plateaus in JOINT’s error on
Foods and Games. This is because JOINT’s error is gap-
dependent while PNF-PEEL and CDP-PEEL’s errors are
more \( k \)-dependent: as \( k \) grows, JOINT’s maximum error
may change as it ranks more items, but the item index where
that error occurs changes monotonically. The reason is that
JOINT’s error ultimately depends on the count gaps under
consideration. In contrast, the item index where PNF-PEEL
and CDP-PEEL incur maximum error may increase and
then decrease. This is because PNF-PEEL and (to a lesser
extent) CDP-PEEL must divide their privacy budget by \( k 
\), and thus are increasingly likely to err (and incur the larger
penalties for) top items as \( k \) becomes large. Figure 2 plots
the maximum error item index and illustrates this effect.

4.3.2. \( \ell_1 \) ERROR

A similar trend holds for \( \ell_1 \) error (Figure 1, rightmost col-
umn). JOINT again largely obtains the best performance
for the Books, Movies, News, and Tweets datasets, with
relatively worse error on Foods and Games. \( \ell_1 \) error is a
slightly more awkward fit for J OINT because JOINT’s utility
function relies on maximum count differences; JOINT thus
applies the same score to sequences where a single item
count has error \( c \) and sequences where every item count
has error \( c \). This means that JOINT selects sequences that
have relatively low maximum (and \( \ell_\infty \)) error but may have
high \( \ell_1 \) error. Nonetheless, we again see that JOINT always
obtains the strongest performance for small \( k \); it matches
PNF-PEEL for small datasets and outperforms it for large
ones; and it often outperforms CDP-PEEL, particularly for
large datasets and moderate \( k \).

4.3.3. TIME COMPARISON

We conclude with a time comparison using the largest
dataset (Foods, \( d \approx 166,000 \)) and 5 trials for each \( k \).
PNF-PEEL uses \( k \) instances of the permute-and-flip mechanism
for an overall runtime of \( O(dk) \). CDP-PEEL’s runtime
is dominated by finding the top-\( k \) values from a set of \( d \) un-
ordered values, which can be done in time \( O(d + k \log(k)) \).
As seen in Figure 3, and as expected from their asymptotic
runtimes, JOINT is slower than PNF-PEEL, and PNF-PEEL
is slower than CDP-PEEL. Nonetheless, JOINT still primar-
ily runs in seconds or, for \( k = 200 \), slightly over 1 minute.

\[ \text{Figure 3. Execution time on a logarithmic y-axis.} \]

5. Conclusion

We defined a joint exponential mechanism for the prob-
lem of differentially private top-\( k \) selection and derived
an algorithm for efficiently sampling from its distribution.
We provided code and experiments demonstrating that our
approach almost always improves on existing pure DP meth-
ods and often improves on existing approximate DP meth-
ods when \( k \) is not large. We focused on the standard setting
where an individual user can contribute to all item counts.
However, if users are restricted to contributing to a single
item, then algorithms that modify item counts via Laplace
noise (Ding et al., 2019; Qiao et al., 2021) are superior to
JOINT and peeling mechanisms. The best approach for the
case where users can contribute to some number of items
larger than 1 but less than \( d \) is potentially a topic for future
work.
Acknowledgements

We thank Ryan Rogers for helpful discussion of the peeling mechanism.

References


Lee, J. and Clifton, C. W. Top-k frequent itemsets via differentially private fp-trees. In Knowledge Discovery and Data Mining (KDD), 2014.

Li, N., Qardaji, W., Su, D., and Cao, J. PrivBasis: Frequent Itemset Mining with Differential Privacy. In Very Large Databases (VLDB), 2012.


A. Proof of JOINT Utility Guarantee (Theorem 3.3)

Proof of Theorem 3.3. The following is a basic utility guarantee for the exponential mechanism.

Lemma A.1 (McSherry & Talwar (2007); Dwork & Roth (2014)). Let \( A(u, D) \) be the utility value produced by an instance of the exponential mechanism with score function \( u \), output space \( R \), dataset \( D \), and optimal utility value \( \text{OPT}_u(D) \). Then

\[
\Pr \left[ A(u, D) \leq \text{OPT}_u(D) - \frac{2\Delta(u)}{\epsilon} \left( \ln(|R|) + t \right) \right] \leq e^{-t}.
\]

Taking \( t = 5 \), and using the fact that \( |R| \leq d^k \) for JOINT’s utility function \( u^* \), completes the result.

B. Full Privacy, Runtime, and Storage Space Proof For JOINT (Theorem 3.9)

Proof of Theorem 3.9. Recall that JOINT refers to the algorithm that uses the efficient sampling mechanism. Here, we first prove that JOINT samples a sequence from the exponential mechanism with utility \( u^* \).

Let EM refer to the naive original construction of the exponential mechanism with utility \( u^* \). It suffices to show that JOINT and EM have identical output distributions. Fix some sequence \( S = (s_1, \ldots, s_k) \) of indices from \([d]\).

If \( s_1, \ldots, s_k \) are not distinct, then JOINT never outputs \( S \). This agrees with the original definition of the exponential mechanism with utility function \( u^* \), which assigns score \( -\infty \) to any sequence of item indices with repetitions. Thus, for any \( S \) with non-distinct elements, \( \Pr_{\text{JOINT}} [\text{output } S] = \Pr_{\text{EM}} [\text{output } S] = 0 \).

If instead \( s_1, \ldots, s_k \) are distinct, by Lemma 3.7, \( \tilde{m}(S) > 0 \). Let \( \tilde{U}_{i^*,j^*} = \min_{i \in [k]} (c_i - c_{s_i}) - z_{s_i} \) be its score in \( \tilde{U} \), so \( i^* = \arg \min_{i \in [k]} (c_i - c_{s_i}) - z_{s_i} \). Let \( U_Z = \{- (c_i - c_j)\}_{i,j \in [d]} \) denote the set of possible values for \( -(c_i - c_j) \); note that this a set of integers and does not have repeated elements. Then

\[
\Pr_{\text{JOINT}} [\text{output } S] = \Pr_{\text{JOINT}} \left[ \text{sample score } \tilde{U}_{i^*,j^*} \right] \cdot \Pr_{\text{JOINT}} \left[ \text{sample sequence } S \mid \text{sample score } \tilde{U}_{i^*,j^*} \right]
\]

\[
= \frac{\tilde{m}(S)}{\sum_{u \in \tilde{U}} \tilde{m}(u) \exp \left( \frac{\varepsilon[\tilde{U}_{i^*,j^*}]}{2} \right)} \cdot \frac{1}{\prod_{r \neq i^*, j^*} \text{exp} \left( \frac{\varepsilon[\tilde{U}_{i^*,j^*}]}{2} \right)}
\]

by Lemma 3.7. Then we continue the chain of equalities as

\[
\frac{\text{exp} \left( \frac{\varepsilon[\tilde{U}_{i^*,j^*}]}{2} \right)}{\sum_{u \in \tilde{U}} \tilde{m}(u) \exp \left( \frac{\varepsilon[u]}{2} \right)} = \frac{\text{exp} \left( \frac{\varepsilon[\tilde{U}_{i^*,j^*}]}{2} \right)}{\sum_{A_{ij}} \sum_{u \in A_{ij}} \tilde{m}(u) \exp \left( \frac{\varepsilon[u]}{2} \right) \text{exp} \left( \frac{\varepsilon[\tilde{U}_{i^*,j^*}]}{2} \right)}
\]

\[
= \frac{\text{exp} \left( \frac{\varepsilon[\tilde{U}_{i^*,j^*}]}{2} \right)}{\sum_{u \in U_Z} m(u) \exp \left( \frac{\varepsilon[u]}{2} \right)} = \frac{m(\tilde{U}_{i^*,j^*}) \exp \left( \frac{\varepsilon[\tilde{U}_{i^*,j^*}]}{2} \right)}{\sum_{u \in U_Z} m(u) \exp \left( \frac{\varepsilon[u]}{2} \right)} \cdot \frac{1}{m(\tilde{U}_{i^*,j^*})}
\]

\[
= \Pr_{\text{EM}} [\text{sample score } [\tilde{U}_{i^*,j^*}]] \cdot \Pr_{\text{EM}} [\text{sample sequence } S \mid \text{sample score } [\tilde{U}_{i^*,j^*}]]
\]

where the second equality uses Lemma 3.5.

Having established the privacy of JOINT, we now turn to proving that its runtime and storage space costs are \( O(dk \log(k) + d \log(d)) \) and \( O(dk) \), respectively.
Referring to Algorithm 1, line 2 takes time $O(d \log(d))$ and space $O(d)$. Line 3 takes time and space $O(dk)$. Line 4 takes time $O(dk \log(k))$ and space $O(dk)$; since each row of $U$ is already decreasing, we can use $k$-way merging (Knuth, 1997) instead of naive sorting.

The loop on Line 8 handles the $\tilde{n}_i$ that are zero. Its variable setup on Lines 5-7 takes time and space $O(k)$. Lines internal to the loop each take $O(1)$ time and space. So, overall, this block of code requires time and space $O(dk)$.

The loop on Line 15 handles the non-zero $\tilde{n}_i$. Its variable setup on Lines 13-14 takes time $O(k)$ and space $O(1)$. Lines internal to the loop each take $O(1)$ time and space. So, overall, this block of code requires time and space $O(dk)$.

Sampling a utility (Line 20) requires time and space $O(dk)$. The remaining loop (Line 22) iterates for $O(k)$ steps, and each step requires $O(d)$ time and space.

Overall, this yields runtime and storage space costs of $O(dk \log(k) + d \log(d))$ and $O(dk)$, respectively.

C. Other Experiment Plots

C.1. Item Count Histograms

Figure 4 contains item count histograms for each of the datasets.

![Item count histograms](image)

*Figure 4. Item count histograms. The x-axis is binned by item count, and the y-axis is the number of items in each bin.*

C.2. $k$-Relative Errors

Figure 5 plots $k$-relative error for each of the mechanisms and datasets. The trends for $k$-relative error are broadly unchanged from $\ell_\infty$ and $\ell_1$ error: JOINT consistently matches or outperforms its pure DP counterpart PNF-PEEL, mostly outperforms CDP-PEEL on large-scale datasets, and is mostly outperformed by CDP-PEEL on small-scale datasets unless $k$ is small. However, $k$-relative error is the least sensitive error (see discussion after Definition 2.2), so for several datasets the performance gaps between methods are small or zero.

C.3. Gamma and Laplace Mechanisms

Figure 6 plots error on the Movies dataset for the core three methods (JOINT, PNF-PEEL, and CDP-PEEL) as well as Gamma (Theorem 4.1, (Steinke & Ullman, 2015)) and Laplace (Ding et al., 2019) mechanisms, which are respectively dominated by PNF-PEEL and CDP-PEEL. Exact details of these mechanisms can be found in the code provided in the supplement. Note that the Laplace mechanism of Qiao et al. (2021) is identical to that of Ding et al. (2019) at the tested value of $\varepsilon = 1$. For $\varepsilon < 0.1$, Qiao et al. (2021) also provides an approximate-DP version of the algorithm, which may be more competitive in that setting.
Figure 5. $k$-relative error plots. Note the logarithmic $y$-axis.

Figure 6. Plots for all error metrics on the Movies dataset, with Gamma and Laplace mechanisms included as additional baselines. Note the logarithmic $y$-axis.